package com.twitter.search.earlybird;

import com.google.common.annotations.VisibleForTesting;

import com.sun.management.OperatingSystemMXBean;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.decider.Decider;

import com.twitter.search.common.decider.SearchDecider;

import com.twitter.search.common.metrics.SearchStatsReceiver;

/\*\*

\* Manages the quality factor for an Earlybird based on CPU usage.

\*/

public class EarlybirdCPUQualityFactor implements QualityFactor {

public static final String ENABLE\_QUALITY\_FACTOR\_DECIDER = "enable\_quality\_factor";

public static final String OVERRIDE\_QUALITY\_FACTOR\_DECIDER = "override\_quality\_factor";

@VisibleForTesting

protected static final double CPU\_USAGE\_THRESHOLD = 0.8;

@VisibleForTesting

protected static final double MAX\_QF\_INCREMENT = 0.5;

@VisibleForTesting

protected static final double MAX\_QF\_DECREMENT = 0.1;

@VisibleForTesting

protected static final double MAX\_CPU\_USAGE = 1.0;

private static final Logger QUALITY\_FACTOR\_LOG =

LoggerFactory.getLogger(EarlybirdCPUQualityFactor.class);

private static final Logger EARLYBIRD\_LOG = LoggerFactory.getLogger(Earlybird.class);

/\*\*

\* Tracks the real, underlying CPU QF value, regardless of the decider enabling

\* it.

\*/

@VisibleForTesting

protected static final String UNDERLYING\_CPU\_QF\_GUAGE = "underlying\_cpu\_quality\_factor";

/\*\*

\* Reports the QF actually used to degrade Earlybirds.

\*/

@VisibleForTesting

protected static final String CPU\_QF\_GUAGE = "cpu\_quality\_factor";

private static final int SAMPLING\_WINDOW\_MILLIS = 60 \* 1000; // one minute

private double qualityFactor = 1;

private double previousQualityFactor = 1;

private final SearchDecider decider;

private final OperatingSystemMXBean operatingSystemMXBean;

public EarlybirdCPUQualityFactor(

Decider decider,

OperatingSystemMXBean operatingSystemMXBean,

SearchStatsReceiver searchStatsReceiver) {

this.decider = new SearchDecider(decider);

this.operatingSystemMXBean = operatingSystemMXBean;

searchStatsReceiver.getCustomGauge(UNDERLYING\_CPU\_QF\_GUAGE, () -> qualityFactor);

searchStatsReceiver.getCustomGauge(CPU\_QF\_GUAGE, this::get);

}

/\*\*

\* Updates the current quality factor based on CPU usage.

\*/

@VisibleForTesting

protected void update() {

previousQualityFactor = qualityFactor;

double cpuUsage = operatingSystemMXBean.getSystemCpuLoad();

if (cpuUsage < CPU\_USAGE\_THRESHOLD) {

double increment =

((CPU\_USAGE\_THRESHOLD - cpuUsage) / CPU\_USAGE\_THRESHOLD) \* MAX\_QF\_INCREMENT;

qualityFactor = Math.min(1, qualityFactor + increment);

} else {

double decrement =

((cpuUsage - CPU\_USAGE\_THRESHOLD) / (MAX\_CPU\_USAGE - CPU\_USAGE\_THRESHOLD))

\* MAX\_QF\_DECREMENT;

qualityFactor = Math.max(0, qualityFactor - decrement);

}

if (!qualityFactorChanged()) {

return;

}

QUALITY\_FACTOR\_LOG.info(

String.format("CPU: %.2f Quality Factor: %.2f", cpuUsage, qualityFactor));

if (!enabled()) {

return;

}

if (degradationBegan()) {

EARLYBIRD\_LOG.info("Service degradation began.");

}

if (degradationEnded()) {

EARLYBIRD\_LOG.info("Service degradation ended.");

}

}

@Override

public double get() {

if (!enabled()) {

return 1;

}

if (isOverridden()) {

return override();

}

return qualityFactor;

}

@Override

public void startUpdates() {

new Thread(() -> {

while (true) {

update();

try {

Thread.sleep(SAMPLING\_WINDOW\_MILLIS);

} catch (InterruptedException e) {

QUALITY\_FACTOR\_LOG.warn(

"Quality factoring thread interrupted during sleep between updates", e);

}

}

}).start();

}

/\*\*

\* Returns true if quality factoring is enabled by the decider.

\* @return

\*/

private boolean enabled() {

return decider != null && decider.isAvailable(ENABLE\_QUALITY\_FACTOR\_DECIDER);

}

/\*\*

\* Returns true if a decider has overridden the quality factor.

\* @return

\*/

private boolean isOverridden() {

return decider != null && decider.getAvailability(OVERRIDE\_QUALITY\_FACTOR\_DECIDER) < 10000.0;

}

/\*\*

\* Returns the override decider value.

\* @return

\*/

private double override() {

return decider == null ? 1 : decider.getAvailability(OVERRIDE\_QUALITY\_FACTOR\_DECIDER) / 10000.0;

}

/\*\*

\* Returns true if the quality factor has changed since the last update.

\* @return

\*/

private boolean qualityFactorChanged() {

return Math.abs(qualityFactor - previousQualityFactor) > 0.01;

}

/\*\*

\* Returns true if we've entered a degraded state.

\* @return

\*/

private boolean degradationBegan() {

return Math.abs(previousQualityFactor - 1.0) < 0.01 && qualityFactor < previousQualityFactor;

}

/\*\*

\* Returns true if we've left the degraded state.

\* @return

\*/

private boolean degradationEnded() {

return Math.abs(qualityFactor - 1.0) < 0.01 && previousQualityFactor < qualityFactor;

}

}